Program Structures & Algorithms

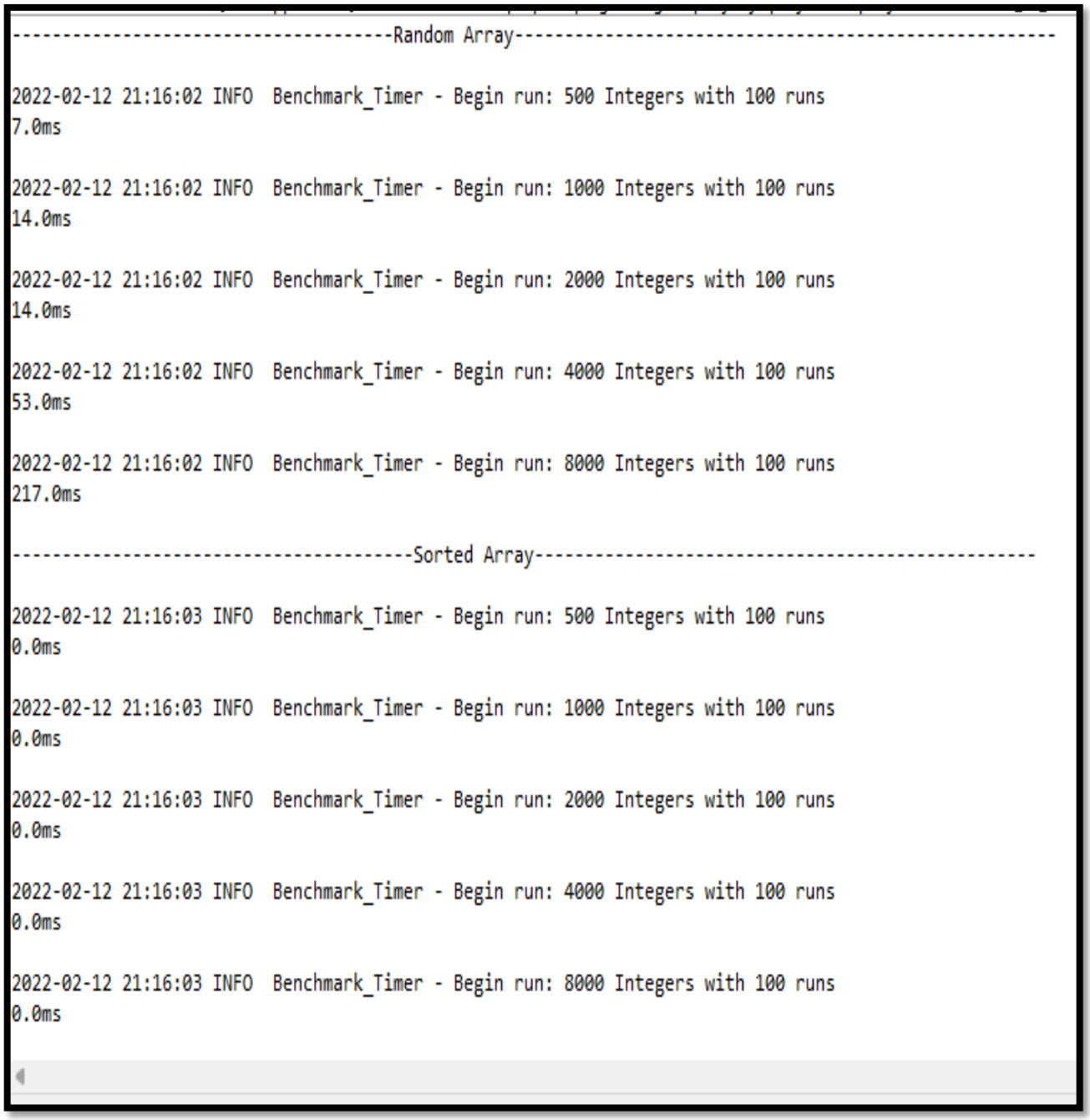
Spring 2022

Assignment No. 2

Name: Shah Divyesh Sureshbhai
NUID: 002922240

- **Task:**
  - Part 1) You are to implement three (3) methods (*repeat*, *getClock*, and *toMillisecs*) of a class called *Timer*. Please see the skeleton class that I created in the repository. *The timer* is invoked from a class called *Benchmark_Timer* which implements the *Benchmark* interface.
  - (Part 2) Implement *InsertionSort* (in the *InsertionSort* class) by simply looking up the insertion code used by *Arrays.sort*. If you have the *instrument = true* setting in *test/resources/config.ini*, then you will need to use the *helper* methods for comparing and swapping (so that they properly count the number of swaps/compares). The easiest is to use the *helper.swapStableConditional* method, continuing if it returns true, otherwise breaking the loop.
    Alternatively, if you are not using instrumenting, then you can write (or copy) your own compare/swap code. Either way, you must run the unit tests in *InsertionSortTest*.
  - (Part 3) Implement a main program (or you could do it via your own unit tests) to actually run the following benchmarks: measure the running times of this sort, using four different initial array ordering situations: random, ordered, partially ordered and reverse-ordered. I suggest that your arrays to be sorted are of type *Integer*. Use the doubling method for choosing *n* and test for at least five values of *n*. Draw any conclusions from your observations regarding the order of growth.

- **Output screenshot**



```
--------------------------------Partially Sorted Array-------------------------------------------

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 500 Integers with 100 runs
0.0ms

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 1000 Integers with 100 runs
3.0ms

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 2000 Integers with 100 runs
13.0ms

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 4000 Integers with 100 runs
55.0ms

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 8000 Integers with 100 runs
214.0ms

-----------------------------------------------Reversed Array-----------------------------------------------

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 500 Integers with 100 runs
1.0ms

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 1000 Integers with 100 runs
6.0ms

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 2000 Integers with 100 runs
26.0ms

2022-02-12 21:16:03 INFO  Benchmark_Timer - Begin run: 4000 Integers with 100 runs
110.0ms

2022-02-12 21:16:04 INFO  Benchmark_Timer - Begin run: 8000 Integers with 100 runs
428.0ms
```

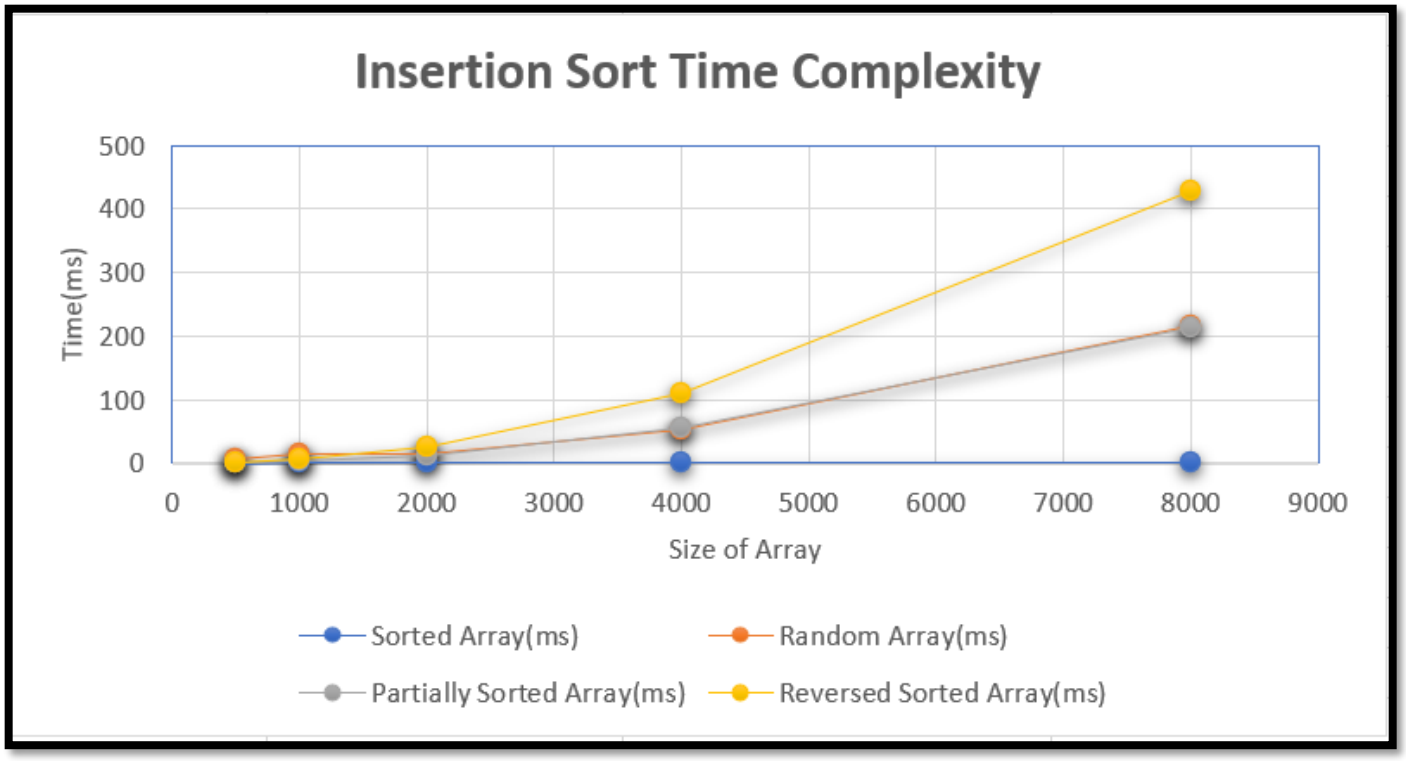
- **Relationship Conclusion**
  - For Partially Sorted, Random Array and Reversed Array, running time increases as number of elements increases but for **Reversed sorted Array** running time increases **exponentially** as size of array increases which is evident from the output above.

  - For sorted array, time complexity is almost $1/100^{th}$ milliseconds which is very negligible irrespective of size of array.

  - Time Complexity:
    - **Sorted Array < Random Array ~ = Partially Sorted Array < Reversed Sorted Array**

- **Evidence / Graph**

N	Sorted Array(ms)	Random Array(ms)	Partially Sorted Array(ms)	Reversed Sorted Array(ms)
500	0.0001	7	0.3	1
1000	0.001	14	3	6
2000	0.002	14	13	26
4000	0.0023	53	55	110
8000	0.0027	217	214	428



- **Unit tests result**